Model Building in Neural Networks and Hidden Markov Models

Michael Wynne- Jones

Doctor of Philosophy

University of Edinburgh

1993

Abstract

This thesis concerns the automatic generation of architectures for neural networks and other pattern recognition models comprising many elements of the same type. The requirement for such models, with automatically determined topology and connectivity, arises from two needs. The first is the need to develop commercial applications of the technology without resorting to laborious trial and error with different network sizes; the second is the need, in large and complex pattern-processing applications such as speech recognition, to optimise the allocation of computing resources for problem solving.

The state of the art in adaptive architectures is reviewed, and a mechanism is proposed for adding new processing elements to models. The scheme is developed in the context of multi-layer perceptron networks, and is linked to the best network-pruning mechanism available using a numerical criterion with construction required at one extreme and pruning at the other.

The construction mechanism does not work in the multi-layer perceptron for which it was developed, owing to the long-range effects occurring in many applications of these networks. It works demonstrably well in density estimation models based on Gaussian mixtures, which are of the same family as the increasingly popular radial basis function networks.

The construction mechanism is applied to the initialization of the density estimators embedded in the states of a hidden Markov model for speaker-independent speech recognition, where it offers a considerable increase in recogniser performance, provided cross-validation is used to prevent over-training.

Declaration

This thesis has been composed by myself and contains original work of my own execution. Some of the work reported in this thesis has been published or reported previously, either as technical reports of commercial institutions, or in the open literature.

Technical Reports

M. Wynne-Jones. Constructive Algorithms and Pruning: Improving the Multi Layer Perceptron. RIPRREP/1000/81/90, Research Initiative in Pattern Recognition, c/o The Librarian, DRA Malvern, St. Andrews Road, Malvern, WR14 3PS, UK, November 1990.

M. Wynne-Jones. A Technique for Dynamically Increasing the Size of the Hidden Layer in a Back-Propagation Network. RIPRREP/1000/82/90, Research Initiative in Pattern Recognition, c/o The Librarian, DRA Malvern, St. Andrews Road, Malvern, WR14 3PS, UK, November 1990.

M. Wynne-Jones. Self-configuring neural networks, a new constructive algorithm, and assessing the importance of individual inputs. Technical Report X2345/1, THORN EMI Central Research Laboratories, Dawley Road, Hayes, Middlesex, UB3 1HH, UK, March 1991.

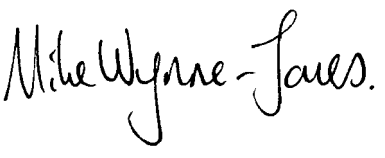
M. Wynne-Jones. Building Gaussian Mixtures in Hidden Markov Models by Component Splitting. CSE1 Research Note 253, DRA Malvern, St. Andrews Road, Malvern, WR14 3PS, UK, August 1993.

Open Literature

M. Wynne-Jones. Constructive Algorithms and Pruning: Improving the Multi Layer Perceptron. Proceedings ***of IMACS '91, the 13th World Congress on Computation and Applied Mathematics, Dublin, Ireland,*** July 1991. Volume 2, pp. 747-750.

M. Wynne-Jones. Node Splitting: A Constructive Algorithm for Feed-Forward Neural Networks. Advances in Neural Information ***Processing Systems 4,*** J. E. Moody, S. J. Hanson and R. P. Lippmann (eds.), pp. 1072-1079. Morgan Kaufmann, 1992.

M. Wynne-Jones. Node Splitting: A Constructive Algorithm for Feed-Forward Neural Networks. Neural Computing and ***Applications,*** volume 1, no. 1, pp. 17-22. Springer Verlag, 1993.



Acknowledgements

The work reported in this thesis was carried out over three and a half years, during which I have benefitted from the help and advice of many people. My principal thanks go to David Bounds, who encouraged me to start the degree in April 1990, when I joined the Research Initiative in Pattern Recognition. I am grateful for his continued support during and after this project, his encouragement and advice, and his support through work at Recognition Research Ltd. and in the Department of Computer Science and Applied Mathematics at Aston University.

I wish to thank my supervisors at the University of Edinburgh, David Wallace and David Willshaw. They showed enthusiasm when I first arrived in Edinburgh to discuss the possibility of an external PhD, and have continued to give me complete support throughout my time there. They have listened to my ideas and helped to shape them, they have given me their complete attention when I have visited Edinburgh, and have had confidence in me even when I was out of touch for months at a time. Thanks to Marcus Frean for early discussions on constructive neural network algorithms, and to Marcus, Nick Woodward, Helen and Martin Holt, Anne Moore and Lesley Smith for putting me up during visits.

I should like to thank Jeremy Severwright and Chris Sharpington of THORN EMI Research, from where I was seconded to the Research Initiative in Pattern Recognition from January 1990 to December 1991. I am grateful for their support in setting up my PhD, and for their help and advice in relating my research to real industrial needs. Thanks also to thank John White of CRL for hardening my resolve.

I am grateful to the staff of the Research Initiative in Pattern Recognition - to . Paul Gregory, Neil McCulloch, Chris Booth, Ian Abraham, Neil Thacker and

John Mayhew for technical advice and cynical appraisal, and to Jacky Bursnell just for being there.

Since the end of RIPR, I have worked on a number of contracts at the Royal Signals and Radar Establishment, now DRA Malvern. I wish to thank Martin Russell for his interest in my research, and for employing me to carry it out. I am grateful to Mark Bedworth, Steve Luttrell, Anthony Heading and David Lowe for help in formulating my ideas, and in finding the mathematics to write them down. John Bridle (now at Dragon Systems inc.) was always interested and helpful, and suggested the idea investigated in chapter 5. I am also grateful to the entire Pattern Processing Group and the Speech Research Unit for their interest in my work, and the use of their office space and computing facilities.

Thanks to Chris Satchwell, Howard James, Peter Herdman, Dave Cressy, Chris Bishop, Justin Wilson and other members of the Neural Computing Applications Forum, who were always interested, and reassured me that it's all worthwhile in the end.

Contents

List of Figures

CHAPTER 1

Introduction

The field of Neural Networks has arisen from diverse sources, ranging from the fascination of mankind with understanding and emulating the human brain, to broader issues of copying human abilities such as speech and the use of language, to the practical commercial, scientific and engineering disciplines of pattern recognition, modelling, and prediction. The pursuit of technology is a strong driving force for researchers, both in academia and industry, in many fields of science and engineering. In neural networks, the excitement of technological progress is supplemented by the evocative and sometimes sinister thrill of reproducing intelligence itself.

The project described in this thesis was initially motivated by a feasibility study entitled 'Guidance Aids for Blind People', a final year electronics project carried out by the author at the University of Southampton in 1989 [121]. In this project, contemporary electronic technology was explored to reproduce experiments from the 1960s [53, 54, 55] and the 1940s [6], in which electronic devices were used to make an audio image of the surroundings. The early work was based on click-echoes, similar to stick-tapping, while the later work reproduced electronically part of the sonar navigation system used by some species of bat. While blind children were able to learn to interpret the sounds produced by the device, the level of concentration required deprived users of more common methods of navigation that were usually employed sub-consciously. The feasibility study concluded that if machine learning techniques could be used to interpret the audio image, offering a much lower-bandwidth, symbolic indication of the surroundings, then progress to a useful guidance aid might be possible.

The second motivation for this study of neural networks was the realisation that while computers are useful tools in problem-solving, traditional methods require a complete breakdown of the problem at hand into a set of problem solving rules, which can then be implemented in a program. In the case of the Guidance Aid, this breakdown is not possible. What are the spectral characteristics of an audio image that enabled users to distinguish the rich texture of a hedge from the hardness of a blank wall, or an object running parallel to the path from a genuine obstacle? Neural networks offer a tool that, when programmed generally, allows data-driven learning of a problem. Given samples of the problem for which a model is required, the neural network implementation can learn the correlations present in the data without heuristic encouragement from the engineer. Given a set of measurements made on a natural system, or an industrial process, neural networks can be used to make symbolic representations of the data, while at the same time making a model of the system itself.

Carried out in an industrial environment, the driving forces of this research were closely related to commercial technological requirements. These included applications in customer modelling, either for classification of customers or for exploration of the structure in a set of measurements made on customer responses, the ability to classify and interpret spoken sounds, and the visualisation of the structure in data through methods of dimensionality reduction. The majority of this thesis, however, addresses the issue of self-configuring neural networks, where the design of an architecture, as well as the setting of internal parameters is driven by the training data, not by the human supervisor. Naturally, if prior knowledge exists regarding the structure of the data, and the characteristics of the generating system, then this should be used to constrain the generic neural network learning tool. If this knowledge is not available, statistical procedures can be used by hand to determine it. Alternatively, fixed-architecture networks are widely available, but they are often slow and can prove unreliable unless applied with great care. Real applications in engineering will only flourish when statistical methods for determining network architectures are automated, making self-configuring neural networks commonplace.

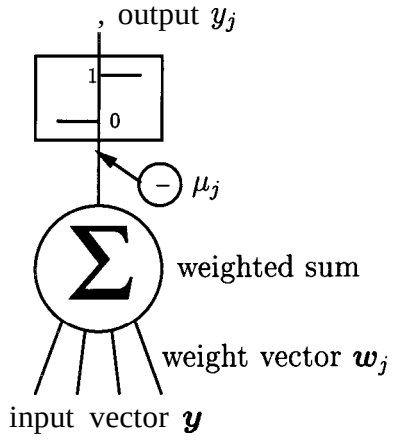


Figure 1.1: McCulloch-Pitts neuron

1.1 Development of neural network structures

A neural network is composed of a number of simple processing units connected by variable strength connections. The processing elements, also referred to as neurons, or nodes, consist of two stages of processing. These are the combination of information from a number of sources, usually other neurons, and the linear or non-linear processing of this activation to provide a single output.

This neuron model was introduced by McCulloch & Pitts [74], who demonstrated that the model could be used to build any finite logical expression. For the McCulloch-Pitts Neuron, the combination of weighted inputs is the sum, and the non-linearity is a threshold function. Formally:

$$Y_j = \begin{cases} 1 & \text{if } \sum_i w_{ij} y_i - \mu_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

Other neuron models are quite widely used, for example in *Radial Basis Function* networks and unsupervised models such as Gaussian mixture density estimators; some of these are explored in detail in chapters 4 and 5.

Networks of McCulloch-Pitts neurons for arbitrary logical expressions were hand-crafted, until the ability to learn by reinforcement of behaviour was developed

in Hebb's book 'The Organisation of Behaviour' [43]. It was proposed that the functionality of neural networks was determined by the strengths of the connections between neurons; Hebb's learning rule proposes that if the pre- and post-synaptic activations are correlated, then the synapse weight is strengthened. This is usually extended from Hebb's case of reinforcement resulting from correlated activation, to a scheme where the connection strength is increased if pre- and post- synaptic activations are both on at the same time, or both off, thereby including reinforcement as a result of correlated non-activation. The effect of this is to increase the probability of repeating desirable network responses if similar inputs are experienced in future, while decreasing the probability of repeating undesirable responses.

1.1.1 Perceptrons

The activation of the McCulloch-Pitts neuron has been generalised to the form

$$y_j = f_j(\sum_i w_{ij} y_i) \tag{1.2}$$

where the activation function, f_j can be any linear or non-linear function. The threshold value, or bias, has been included in the sum in this expression, with the assumption of an extra component in the $\boldsymbol{y}$ vector whose value is fixed at 1. Rosenblatt studied the capabilities of groups of neurons in a single layer, and hence all acting on the same input vectors; this structure was termed the Perceptron [100], and Rosenblatt proposed the Perceptron Learning Rule for learning suitable weights for classification problems [101]. Equation (1.2) defines a non-linear function across a hyperplane in the input space; with a threshold activation function the neuron output is simply 1 on one side of the hyperplane and 0 on the other. When combined in a perceptron structure, neurons can segment the input space into regions, and this forms the basis of the capability of neural networks to perform classification.

Minsky and Papert pointed out, however, that many real world problems do not fall into this simple framework, with the infamous example of the XOR function of two variables. Here it is necessary to join together two regions of the input space,

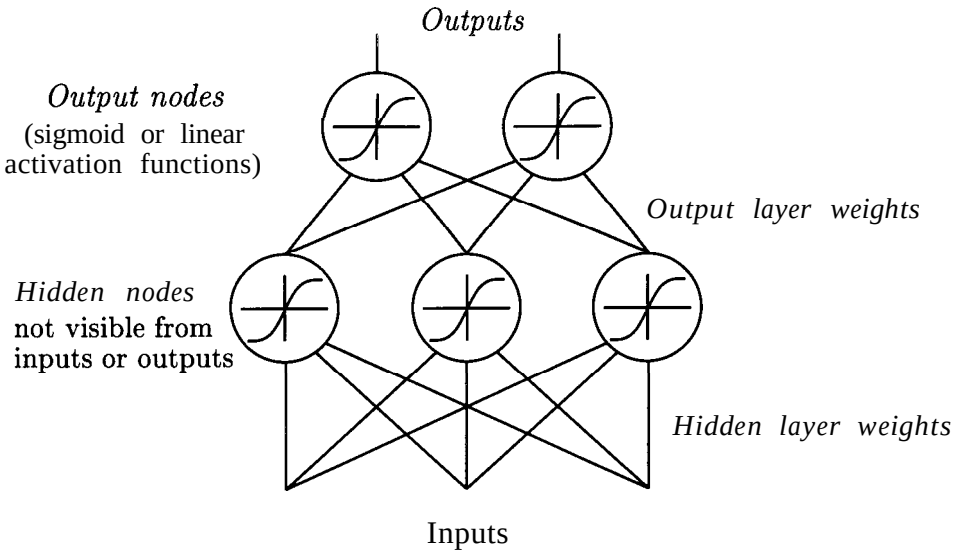


Figure 1.2: Multi-layer perceptron structure and terminology

and they showed that this was possible with a two-layer perceptron structure [76]. This formed the *Multi-layer Perceptron* (MLP) which is widely in use today, although the Perceptron Learning Rule (also called the delta rule) could not be generalised to find weights for this structure. The multi-layer perceptron structure and terminology are shown in in figure (1.2).

A learning rule was proposed in 1985 which allows the multi-layer perceptron to learn. This generalised delta rule is based on the back-propagation of error derivatives through the network [102, 103, 118]. The publication of this learning rule initiated the recent academic interest in neural networks, and the field subsequently came to the attention of industrial users. This has resulted in a large number of academic publications and successful industrial applications.

1.1.2 Network optimization in multi-layer perceptrons

On presentation of data to a network, if the desired response is produced, then the network parameters should be modified to increase the probability of producing a similar response to similar patterns in the future. Conversely, if an undesirable response occurs, the network parameters should be modified to make a re-occurrence less likely. This leads to optimization of the network weights with respect to an error function, measuring the fit of the model stored in the network

to the training data.

Given a set of training vectors $\boldsymbol{x}_p$, and a corresponding set of preferred or target output vectors $\boldsymbol{t}_p$, which the network outputs $\boldsymbol{y}_p$ aim to match, the error function is commonly proportional to the sum of the squares of the errors for individual patterns, in line with statistical modelling techniques. Minimization of the sum square error with respect to the free parameters in the network leads the network to approximate the Bayes discriminant vector, the probability of a class given the input to the network [15, 66]. Hence:

$$E = \frac{1}{2}\sum_{\mathrm{p}}(\boldsymbol{t}_p - \boldsymbol{y}_p)^2 \tag{1.3}$$

The most common form of optimisation used in multi layer perceptrons is gradient descent. The weights are adjusted a little at a time in a suitable direction to decrease the error; the direction and magnitude are determined from the derivative of the error with respect to each weight.

$$\Delta w_{ij} = -\epsilon\frac{\partial E}{\partial w_{ij}} \tag{1.4}$$

The error gradient $\frac{\partial E}{\partial w_{ij}}$ cannot be calculated immediately from the error function in equation (1.3), and so the chain rule for partial derivatives is used to find it. This relies on all parameters of the network being differentiable, including specifically the activation functions $f(x_j)$ where $x_j = \sum_i w_{ij}y_i$ for units j in one layer fed from units i in the previous layer. The threshold activation function does not comply with this criterion, but the more common linear function and the sigmoid, tanh, or smoothed threshold functions do.

For the output layer of the network, we have

$$E = \frac{1}{2}\sum_{p}\sum_{l}(y_{jp} - t_{jp})^2, \tag{1.5}$$

and the derivative of the error with the respect to the output of an individual unit (y_j) is:

$$\frac{\partial E}{\partial y_j} = \sum_{\mathbf{P}} (y_{jp} - t_{jp}) \,. \tag{1.6}$$

For a network of elements with weighted sum activations $x_j = \sum_i w_{ij} y_i$, and smooth output activation functions $y_j(x_j)$, derivatives of the network error with respect to these quantities can be found. $\frac{dy_j}{dx_j}$ depends on the function $y_j(x_j)$; $\frac{\partial x_j}{\partial y_i} = w_{ij}$ and $\frac{\partial x_j}{\partial w_{ij}} = y_i$.

The chain rule allows us to find the derivative of the error with respect to x_j:

$$\frac{\partial E}{\partial x_j} = \sum_p \frac{\partial E}{\partial y_{jp}} \frac{dy_{jp}}{dx_{jp}}, \tag{1.7}$$

and a second application of the chain rule gives an expression for the error -weight gradients needed for the weight update:

$$\frac{\partial E}{\partial w_{ij}} = \sum_p \frac{\partial E}{\partial y_{jp}} \frac{dy_{jp}}{dx_{jp}} \frac{\partial x_{jp}}{\partial w_{ij}} \tag{1.8}$$

$$= \sum_p \frac{\partial E}{\partial y_{jp}} \frac{dy_{jp}}{dx_{jp}} y_{ip}. \tag{1.9}$$

An equivalent application of the chain rule gives the derivative of the error with respect to the outputs of the previous layer, and this expression is the equivalent of equation (1.6) for layer i:

$$\frac{\partial E}{\partial y_i} = \sum_p \frac{\partial E}{\partial y_{jp}} \frac{dy_{jp}}{dx_{jp}} \frac{\partial x_{jp}}{\partial y_{ip}} \tag{1.10}$$

$$= \sum_p \frac{\partial E}{\partial y_{jp}} \frac{dy_{jp}}{dx_{jp}} w_{ij}. \tag{1.11}$$

Repeated use of equation (1.9) to find error-weight derivatives, followed by equation (1.11) to find the error-output derivative at the previous layer, allows updates to be calculated for all weights in the network.

Repeated use of gradient descent using the method of back-propagation of error derivatives eventually brings the network to a local minimum of error with respect to the weights given the training data, although for non-linear activation

functions there is no guarantee that a global minimum will be reached. The chances of reaching a global minimum are enhanced by the use of *momentum* in training, where the weight update depends on the previous update in addition to the local gradient:

$$\Delta w_{ij}^{(n)} = -\epsilon \frac{\partial E}{\partial w_{ij}}^{(n)} + \alpha \Delta w_{ij}^{(n-1)} \tag{1.12}$$

The momentum term encourages the trajectory in weight space to continue with its direction and speed, being changed only gradually by local gradients. This speeds convergence of the optimization process, and helps to prevent it from becoming stuck in local minima.

Second-order optimization methods have been used, where weight updates are calculated using a locally quadratic model of the error surface rather than the locally linear model assumed in gradient descent. Optimization is achieved by repeatedly moving to the estimated position of the parabola's minimum [4, 117].

1.2 Thesis summary

This thesis is devoted to the issue of self-configuring neural networks, where the design of the architecture, as well as the setting of the internal parameters, is driven by the training data, not the human supervisor. Chapter 2 reviews schemes for modifying network architectures automatically, mainly in the context of layered networks following the Perceptron and Multi-layer Perceptron formalisms, and this review leads to the identification of a number of requirements for a new constructive algorithm.

These requirements are formalised in chapter 3 as a set of axioms for the design of architecture-modifying algorithms, and a scheme is proposed for adding new nodes to a network by splitting an existing node in two. This technique is tested in two simple test problems, where it appears to work.

The latter part of chapter 3 moves away from the mechanism for splitting a node, towards the identification of which nodes in a network are the best candidates for splitting. In line with the axioms set out at the beginning of the chapter, the

criterion for selecting nodes for splitting is integrated with the best criterion for on-line removal of useless nodes (pruning), so that at one end of the scale nodes are irrelevant and can be pruned, while at the other they benefit from splitting in two.

The techniques developed in the chapter are then thoroughly tested on a number of classification problems, with the result that they do not, in fact, yield the expected benefit in network performance. The reason for this is identified, namely that while the techniques are viable in general, they are not suitable for use in models with long-range effects, such as the multi-layer perceptron.

The axioms for development of a constructive algorithm are reiterated in chapter 4, where the construction of a model by splitting its constituent parts is implemented in the context of a Gaussian mixture model for maximum-likelihood density estimation. This scheme is demonstrated for a synthetic data set, and for a real data set from speech recognition.

The criteria that must be satisfied by a model for component-splitting to be applicable are specified at the end of chapter 4, and constructive schemes that already exist for neural network models satisfying these criteria are briefly reviewed. The possibility of a constructive approach based on splitting model components is briefly discussed in the context of the topographic-mapping Kohonen [57, 58] and elastic networks [23, 22], and is suggested and demonstrated diagrammatically for Finite Element Analysis models.

Following the successful demonstration of component-splitting in Gaussian mixture models for density estimation, the technique is applied in chapter 5 to the density models within the states of a hidden Markov model for speech recognition. Results show that models with mixture-components allocated by node-splitting outperform equivalent sized models with uniform component allocation.

The findings of the thesis, which have been developed over the last three years, are discussed in chapter 6, against the background of the changes that have taken place in academic circles and industrial use of neural networks over the same period.

CHAPTER 2

Constructive Algorithms and Pruning: A Review

This chapter reviews a number of techniques that have emerged recently, which attempt to improve on the perceptron and multi-layer perceptron training algorithms [100, 102, 103, 118], by changing the network architecture as training proceeds. These include pruning useless nodes or weights, and constructive algorithms where extra nodes are added as required. The advantages include smaller networks, faster training times on serial computers, and increased generalization ability, with a consequent immunity to noise. In addition, it is often much easier to interpret what the trained network is doing. One can then begin to draw analogies with other pattern classifying techniques such as decision trees and expert systems. A number of architecture-modifying schemes are reviewed in this chapter, and suggestions are made regarding the classes of problem to which they are most applicable. Conclusions are drawn which helped to shape the work presented in later chapters.

2.1 Constructive algorithms and pruning

Multi-layer perceptron networks are well established as a standard neural network technique for pattern recognition tasks. They are hampered, however, by a number of problems and arbitrary parameters which make them much less dependable and predictable than they could otherwise be. The main problem is that there is no way of determining in advance how many units there should

be in the hidden layer. If there are too few, the network may not learn at all, while too many hidden nodes lead to over-learning of individual samples in the training data, at the expense of the formation of an optimal model of the statistical distributions underlying the training data. This leads to an inability to generalise, so that previously unseen measurements are labelled according to the nearest training sample, rather than in accordance with a good model of the problem. Despite numerous analytical and heuristic attempts to determine this number [38, 63, 64, 77], general and reliable methods have only recently begun to emerge. The problem of generalization as a function of model size is demonstrated in figure 2.1.

Recent developments fall into two categories. Pruning algorithms build and train large networks, and then remove nodes or weights that contribute little to the network's operation, while constructive algorithms attempt to form an approximate solution using a small network, and then add further nodes to increase the precision as required. It has been demonstrated [107, 108] that a larger network is generally required to learn a classification task, than is needed simply to implement a known solution using pre-determined weights. To this end, an ideal algorithm for determining network architecture might add new nodes during early training, and apply pruning selectively or inherently as a solution is approached. The need for additional nodes to enable the learning of a solution arises because a network of the minimal size required to solve a problem is likely to get stuck in one of many locally optimal solutions in the training phase. Larger networks also have locally optimal solutions, but can have a large number of equally good global minima. The probability of finding a good solution is increased with larger networks, and pruning algorithms rely on the global minimum being retained through the pruning event. In constructive algorithms, we begin with a network insufficiently large to exhibit many local minima in addition to the smooth global minimum of an approximate solution, and hope that the minimum found will remain global, and be deepened by the addition of new nodes. To work well, parameters associated with newly added nodes must have pre-determined values so that the new network is at least as good as its parent.

By pruning weights (or nodes) that do not contribute significantly to the classification or mapping task, we hope to reduce to a minimum the number of degrees

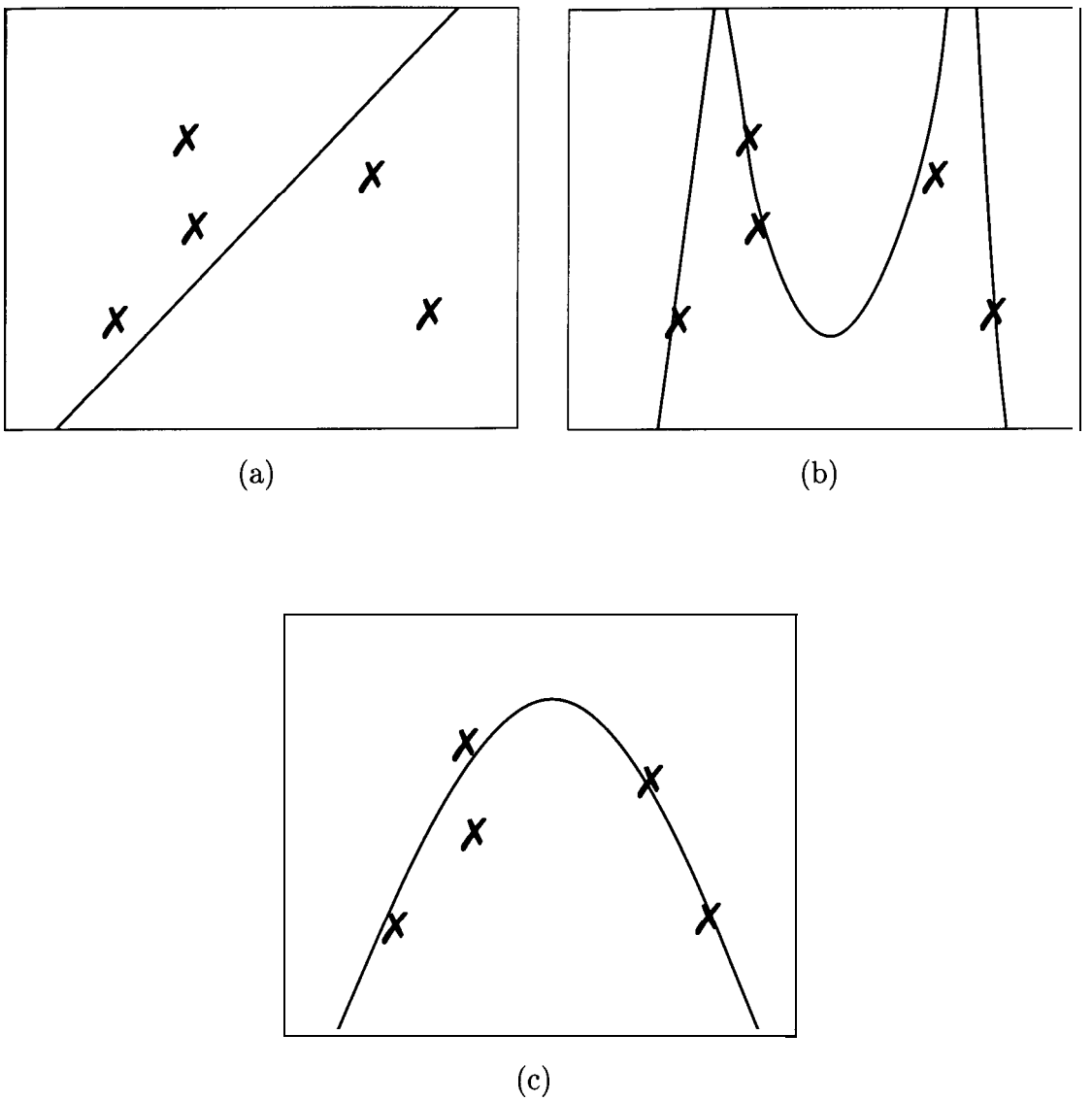


Figure 2.1: Generalization ability as a function of model size. The data points could have been generated by one of many systems. An over-simple model a) learns only a very approximate fit to the data, while an over-powerful mo 6el learns the individual samples, but can behave unpredictably in other parts of the space (b). An appropriately sized model passes close to the data points, without necessarily passing through them, but is a more accurate model of the system which generated the data (c).

of freedom used by the network to implement the solution, thereby ensuring we have a simple model of the system. One way of making this obvious implementation of Occam's Razor is to minimise the total activation in the hidden layer, which encourages the nodes there to act orthogonally [18]. This is the same behaviour for which we aim in inductive inference, namely in decision trees and automatically generated expert systems. It may be possible to extract information directly from the weights of such a network, allowing us to determine the

problem-solving rules that have been learned. Naturally the reduction of the number of nodes to a minimum detracts from the fault tolerance usually associated with neural networks, and so some thought is needed before such minimalist techniques are used for a particular application.

2.2 Constructive algorithms

Early constructive algorithms built multi-layer feed-forward networks of perceptron units [100], which could be applied to problems involving binary input patterns. Convergence of such a network is guaranteed if the training data is linearly separable [12]. The Pocket Algorithm [32, 33, 35] is an extension of the Perceptron Learning rule [100], that allows approximate solutions to be found for non-linearly separable problems, by holding 'in one's pocket' the best set of weights found so far. A faster-learning and more robust algorithm is proposed in the thermal perceptron [30], where perceptrons initially form soft decision boundaries, which become harder as training proceeds according to an annealing schedule. For binary patterns, it is always possible to cut off a corner of the binary hypercube with a plane, thereby ensuring that we can learn a binary function of binary patterns by repeatedly adding perceptrons to a network. These techniques have been implemented in Gallant's Tower, Inverted Pyramid, and Distributed algorithms [34, 35] and the Tiling Algorithm [75, 83], although these have largely been superseded by the Upstart Algorithm [29, 28], which is more efficient in terms of the number of nodes created. They mostly build tree-like networks with finer resolution of the input space towards the leaves of the tree, and do not usually include a stopping criterion to halt the addition of new nodes or layers. This means that every sample in the training set is learned, but has strong repercussions if the training set is incomplete, has noise, or is derived from a classification problem where the class distributions overlap. These networks are good for learning completely a logical data set, but are likely to fail to find a useful model of many statistical distributions.

Later methods [2, 3, 27, 39, 97, 111, 122], reviewed in the following sections, apply to non-binary functions of non-binary inputs. They usually build a single hidden layer, which has an advantage over the 'deep network' methods that the

propagation time for data from the input to the output is shorter, and constant. This is particularly important for real time signal processing applications, but not usually in simulated networks used for classification. They do not guarantee to learn every sample in the training set, but are more likely than the earlier algorithms to converge to solutions with good generalization ability for statistical problems.

2.2.1 Tiling algorithm and Upstart algorithm

The Tiling Algorithm [75, 83] builds a layered network, with each layer approximating the solution of a binary classification problem more closely than the previous one. A master unit in each layer learns at least one new pattern, and the ancillary units ensure that no previously learned patterns are lost. The algorithm creates only as many nodes as it needs to build a solution, and it has been demonstrated to have good generalization capabilities. On the down side, it requires an excessively large number of nodes, as most ancillary units just duplicate the action of those in the previous layer. This problem is avoided in later methods (Upstart, Cascade correlation), by allowing connections to span many layers. A recent method that has grown out of the Tiling algorithm is the Neural Tree, [110] and other similar tree algorithms [36, 72]; these make repeated use of the Pocket algorithm to divide the input space into partitions for classification, and build a decision tree [14] in the process. These techniques look promising, but comparative results with other constructive algorithms have not been published.

The Upstart algorithm [29, 28] builds a binary tree of nodes. For each node, one subtree corrects all errors where a one is expected, and the other corrects all the errors where a zero is expected. Each sub-node is guaranteed to classify at least one of its targets correctly, and so convergence is guaranteed for the training data. The number of nodes grows linearly with the number of training patterns for the theoretically hardest problem, a random boolean function, and this is better than the node growth rate of the Tiling algorithm. Training can be speeded up if each subtree is trained using only the patterns that have not already been learned elsewhere in the tree. Extensions are possible whereby

the trained network is mapped directly onto a feed-forward network with only one hidden layer. The Upstart algorithm is easily extendible to classifiers with several possible output classes and real-valued inputs, while preserving many of its advantages. Knerr's constructive algorithm [56] is similar to Upstart, building a tree of neurons with each branch correcting some errors of the parent tree's classification.

2.2.2 Dynamic node creation in standard MLPs

Ash [2, 3] describes a system where the error rate is analysed as training proceeds, and a new hidden node is added whenever the error rate is no longer decreasing significantly, but is not acceptably low. The new nodes are introduced with small random weights, and the results are encouraging when compared with the training times required by standard networks of fixed size. The scheme includes an analysis of when new nodes should be added, based on the assumption that the error rate decreases exponentially as training proceeds, and this scheme is built on in section 3.6. It should be possible to find a better way for determining the initial weights of the new nodes, as small random weights do not guarantee that the larger network is at least as good as the old one. A scheme with similar ad-hoc initialisation is described in [46].

An alternative technique [111, 97] adds new nodes to the hidden layer, but speeds up training by freezing the weights of the previous ones. Back-propagation's property of attempting to force each hidden node to account individually for all the error in the output layer means that the newly introduced nodes learn the errors of the previous network with frozen weights. Interestingly, this feature is usually considered a disadvantage, as the use of teamwork rather than individualism amongst nodes might be expected to find a solution more quickly.

2.2.3 Cascade correlation

In this algorithm, which applies to real-valued inputs and outputs, hidden units are added with inputs from all previous input and hidden units [27]. The weights to all previous units are frozen, and the new unit learns a mapping which has the

best possible correlation with the errors of the previous network. The weights to the new unit are then frozen, and the process continues until there are no more errors. This method is not limited to binary classifications, although published results only cover these problems. There is no back-propagation, with the result that the networks train quickly; the speed compares very favourably with both backprop and quickprop (see section 1.1.2) [25, 26] when learning to separate two interlocked spirals, In a thorough evaluation of cascade correlation in comparison with standard MLPs using data from four real problems of different complexities by Indurkhya and Weiss [49], it is apparent that cascade correlation learns faster, but yields classifiers with lower cross-validation performance. The numbers of hidden units required for modelling these problems were similar for both networks, leading Indurkhya and Weiss to suggest the use of cascade correlation to determine an appropriate network size, which should then be trained by standard techniques.

The problem of learning individual samples from overlapping distributions might be addressed by continuing the development of the network until the error figure found by correlation decreases to the value expected from prior knowledge of the problem. This could be a noise model, or a measure of the degree to which the class distributions are known to overlap. Since cascade correlation builds tall networks, with many layers and connectivity from each layer to all earlier layers, it has the advantage of enhanced feature detection over standard networks with few layers. On the other hand, the many-layered architecture leads to a variable delay between application of an input and the appearance of the result.

2.2.4 Meiosis networks

Meiosis networks [39] arise from an approach to multi-layer perceptrons which combines the usual gradient descent optimization with a stochastic search. This has the advantage that it is possible for the network to escape from a local minimum by a random perturbation in the weight space, with a probability that decreases as the network approaches a good solution. This is implemented using stochastic weights, sampled from Gaussian distributions each time a weight value is required. The learning algorithm updates the means and the standard

deviations of the weight distributions. There are three learning rules for the iteration at time n: the mean $\mu_{w_{ij}}$ is updated according to the normal learning rule described in section 1.1.2 above:

$$\mu_{w_{ij}}{}^{(n+1)} = \alpha\left(-\frac{\partial E}{\partial w_{ij}^*}\right) + \mu_{w_{ij}}{}^{(n)}, \tag{2.1}$$

where w_{ij}^* is the weight value drawn from the Gaussian weight distribution. The standard deviation $\sigma_{w_{ij}}$ is increased at each update to reflect the uncertainty indicated by a large update:

$$\sigma_{w_{ij}}{}^{(n+1)} = \beta\left|-\frac{\partial E}{\partial w_{ij}^*}\right| + \sigma_{w_{ij}}{}^{(n)}. \tag{2.2}$$

A third learning rule determines the decay of the standrd deviation with time, allowing the weights eventually to become deterministic:'

$$\sigma_{w_{ij}}{}^{(n+1)} = \zeta\sigma_{w_{ij}}{}^{(n)}; \quad \zeta < 1. \tag{2.3}$$

Meiosis is the process of one node splitting to create two new ones. The *composite* variance is computed for each node in the hidden layer, and the split occurs for any node whose composite variance, that is the standard deviation relative to the mean, is greater than 100% for weights both into and out of the node.

$$\frac{\sum_i \sigma_{ij}}{\sum_i \mu_{ij}} > 1 \text{ and } \frac{\sum_k \sigma_{jk}}{\sum_k \mu_{jk}} > 1 \tag{2.4}$$

The mean of each new weight is a jittered copy of the original, perturbed in random directions by a small proportion of the standard deviation in the chosen direction, and each has half the variance of the old weight distribution. This kind of splitting policy has the advantage that it does not converge to a complete fit of the training data, and consequently the resulting networks are likely to exhibit better generalization than the ones produced by the Pocket, Tiling, and Upstart algorithms. The decision on whether to split is made using only locally measured

'It is not clear in [39] how equations (2.2) and (2.3), which propose different updates to $\sigma_{w_{ij}}$, are implemented in the same system.

parameters, making the scheme attractive in terms of biological plausibility and distributed computing architectures.

2.2.5 Summary: What to do next with constructive algorithms.

The constructive algorithms discussed in this section all lack a good mechanism for determining the weights of a newly added node, although Meiosis is good in the context of stochastic weights. Accepting the idea of adding new nodes by splitting an old node in two, we also require a good measure of the degree to which a given node requires splitting. These requirements are explored in chapter 3, which also builds on ideas from pruning, to obtain an ordered list of nodes with the most prunable at one end, and the most splittable at the other.

2.3 Pruning

Pruning has been carried out on networks in three distinct ways. The first is a heuristic approach, identifying which nodes and weights contribute little to the mapping. After these have been removed, additional training leads to a better network than the original. An alternative technique is to include terms in the error function, so that the weights tend to zero under certain circumstances. Zero weights can be removed without degrading the performance of the network. Finally, if we define the sensitivity of the global network error to the removal of a weight or node, and evaluate it for each such parameter in the network, we can then remove the weights or nodes to which the global error is least sensitive. The sensitivity measurement does not interfere with training, and involves only a small amount of extra computational effort. It is well matched for implementation alongside the construction scheme mentioned in section 2.2.5, and developed in chapter 3.

2.3.1 Heuristic pruning

Sietsma and Dow have described a pruning scheme based on analysing automatically the activations of nodes, in response to different patterns in the training data [108, 107]. Their pruning takes place in three different ways:

1. Pruning of units that do nothing, or duplicate the action of other units. A unit is redundant if it has the same output for all patterns in the training data, or duplicates the action of another node.

2. Pruning of units that provide unnecessary information. If a node makes a distinction between patterns that are later recombined, then the node can be deleted.

3. Pruning out an entire network layer. A network with many layers may well have one or more of them redundant, especially after the earlier stages of pruning have been carried out. Indeed, it has been shown [31, 47] that one hidden layer is theoretically sufficient to implement any problem, although this may require more nodes than a multi-layer network.

2.3.2 Pruning which is inherent in the learning algorithm

A lower-level approach is more appealing than the heuristic pruning scheme, as the constraints on structure for the network come from the network itself, rather than from a global monitoring system. On the other hand, systems can be envisaged where processes such as adding nodes and pruning would be carried out by exactly this kind of global control process, which could itself be a rule based system or a neural network. Current on-line pruning techniques either minimise a biased cost function, imposing constraints on the final weight values, or halt training periodically to measure the prunability of the individual weights and nodes, pruning those with the highest values.

Minimising a biased cost function

In standard learning techniques we optimise a network in terms of a cost function $O = E$ that describes the goodness of fit of the model stored in the network, to the data representing the problem to be learned. A second term B can be added to the cost function (equation 2.5) incorporating prior knowledge of the problem, such as the kind of architecture that is likely to be most useful, constraints on weight values, or constraints on the function implemented by the network.

$$O = E + B \tag{2.5}$$

A constraint on the function implemented might be that the network should not learn to classify individual samples from overlapping distributions; elimination of such unlikely solutions leads to the extra term being referred to as a bias or regularising term in line with the field of non-parametric statistics [15, 93, 92, 114].

In relation to pruning networks, or the production of minimal architectures, the constraints we wish to apply by means of the bias term will be such that they minimise the number of active nodes or weights in the network. These can be removed from the network before training is continued.

Rumelhart's pruning mechanism (unpublished) involves a bias term of

$$\sum_i \sum_j w_{ij}^2 \tag{2.6}$$

giving a modified weight update rule

$$\Delta w_{ij}{}^{(n+1)} = -\epsilon \frac{\partial E}{\partial w_{ij}}^{(n)} - \beta w_{ij}{}^{(n)} \tag{2.7}$$

where $\beta < 1$. This causes the weights to decay exponentially towards zero, and weights that come sufficiently close to zero can be pruned out. This approach was successfully used in studies on network generalization by Hinton [45] and Thodberg [113], and it has been analysed in mathematical detail by Krogh and

Hertz [59]. This work was taken a step further by Hanson and Pratt [40], and Rumelhart, with the aim that high and low weights should decay strongly, while mid-range weights are left unaffected. High weights are discouraged for reasons of smoothness, while low weights can be pruned. Nowlan and Hinton [85] generalised this principle, discouraging a large number of different mid-range weights by constraining the weights to a Gaussian mixture distribution with few components. The mixture parameters were optimised at the same time as the weights themselves. This procedure does not rule out any weight values in favour of others, but constrains the network by clustering the weights to a small number of groups of similar values. The number of free parameters in the network is greatly reduced, and generalization performance is, once more, enhanced.

Hanson and Pratt's experiments showed that different bias terms are indeed useful for finding minimal architectures, but the bias term could not easily be used in conjunction with the momentum term used traditionally for avoiding local minima. Nowlan and Hinton use conjugate gradient optimization, which does not employ momentum. It may be possible to combine the clustered weights of Nowlan and Hinton [85] and Hanson [39], to make a network with the number of weight clusters determined by a constructive approach such as node-splitting, introduced in the chapter 3.

Biased cost functions have been used in other applications too, first for pruning nodes by including a node relevance term in the cost function [19]. The relevance term is defined as a product of functions of the weights into and out of a node, and minimising the cost function then minimises the total number of relevant nodes. This kind of node decay (cf. weight decay, above) has not been analysed in detail, but gives promising initial results. A second example encourages the optimal use of hidden units [18], by including the hidden layer activations in the cost function to be minimised, and hence forcing the units to find a non-linear, but locally orthonormal basis set that spans a subspace of the input space. This attempts to avoid unnecessary units right from the start of training, rather than allowing them to develop and eliminating them later. This technique could prove useful in the context of integrating neural networks with expert systems or decision trees, since these systems attempt to invoke independent rules whenever possible. An alternative use is in dimensionality reduction for data visualisation and efficient

coding. A similar technique based on non-linear principal components has been suggested by Webb [116], and other techniques have been investigated by the author [unpublished] based on Kohonen networks [57] and elastic networks [23].

The initial aim of Rumelhart's use of the biased cost function was to avoid large weights, and consequently to ensure smoothness of the mapping implemented by a network. A sigmoid transfer function is linear for small input values, but approximates to a step for larger values (and hence for large weights). Step functions are necessary to classify individual samples from a statistical distribution, and it has already been emphasised that this should be avoided if at all possible.

An alternative approach for discouraging the fitting of individual samples, also enforcing a low-curvature mapping, is to include a term in the cost function representing the curvature of the mapping, averaged over the input domain [7, 8, 9]. Eliminating high curvature eliminates sharp transitions, and has a similar effect to the bias term in equation 2.5. If the data distributions are smooth, and the curvature of the mapping is minimised, the performance is likely to be improved for the classification of previously unseen patterns. Minimal networks implementing low-curvature mappings would also be more likely to perform usefully in extrapolation, that is for the classification of patterns outside the input domain represented by the training data.

2.3.3 Pruning according to an ordered list of node or weight relevances

Mozer and Smolensky [80, 81, 82] opt for an all-or-none approach to pruning, rather than gradual pruning by means of weight decay. They investigate the pruning sensitivity, or relevance of a unit in the network, defined as:

$$\rho_j = E_{\text{without unit } j} - E_{\text{with unit } j} \tag{2.8}$$

so that the relevance of a unit depends on how much the network global error will increase if the node is removed. Pruning could then be applied for the least relevant units. The relevance measure is determined by the shape of the error surface around the minimum to which the network has been trained. Mozer and

Smolensky identify a problem in measuring the shape of the local minimum just from the gradient information calculated by standard back-propagation, since the gradient falls away to zero near the minimum. They repeat the back-propagation of errors using a modulus linear error function:

$$E = \sum_{\mathrm{P}} \sum_{j} |t_{pj} - y_{pj}| \tag{2.9}$$

whose gradient does not disappear to zero near the minimum. The minimum of this function, however, is not necessarily in the same place, as deep, or of the same steepness as the sum square error minimum.

A better approximation to the shape of the minimum, which maintains the sum square error function, is to use the first three terms of the Taylor series, namely:

$$E(a) = E(a_0) + \frac{\partial E(a)}{\partial a}\bigg|_{a=a_0} (a - a_0) + \frac{1}{2} \frac{\partial^2 E(a)}{\partial a^2}\bigg|_{a=a_0} (a - a_0)^2 \tag{2.10}$$

The first term can be ignored as it does not describe the shape of the minimum; the second term decreases to zero near the minimum. The third term measures the sharpness of the local minimum, and hence the sensitivity to pruning, and higher order terms are assumed to be negligible. The second derivatives of the sum squared error with respect to network parameters can be found by back-propagation [42, 62] or by measurement [95], although the measurement technique is not suitable at local minima (see section 3.5).

The question of what parameter of a node the shape of the error surface should be measured against is resolved by introducing the attentional strength, α_i of a unit, where the output of a unit j is now $y_j = f_j(\sum_i w_{ij}\alpha_i y_i)$ and the effect of removing unit i is expressed as $\rho_i = E_{(\alpha_i=0)} - E_{(\alpha_i=1)}$. This scheme is discussed in more detail in section 3.5. Relevance of a parameter, for more general puropses than removal of the parameter, is determined by the shape of the error surface around $\alpha = 1$. This pruning technique leads to networks from which rules could be extracted quite easily for low dimensional boolean problems, and the relevance measure facilitated an evaluation of which rules were the most frequently invoked in a given problem, that is, which splits in the feature space were the rules, and

which are the exceptions. This scheme has also been applied to the network input layer, to identify which inputs are relevant for a classification problem and which are not [80, 81, 82, 123].

Le Cun, Denker and co-workers have carried out similar experiments for pruning individual weights [62], using the second derivative term from the Taylor series, and the results are very promising. The number of free parameters in the networks used to implement hand-written digit recognition was reduced by a factor of four by the use of this pruning scheme [61]. There are no results available for comparisons between this pruning method and those discussed earlier, but this is the most convincing way of identifying which nodes are to be pruned. The idea of an ordered list of sensitivities of the network global error to each weight was used also by Karnin [52]. According to his scheme, the sensitivity would ideally be determined by integration over the entire weight space, but since this is not possible, (c.f. training by exhaustive search), it is integrated just along the training path. It seems likely that the sensitivity found in this way would depend most strongly on the starting weights (which are random) and not be as good as the measures based on characteristics of local minima described above.

2.4 Discussion

We have reviewed a number of techniques that aim to build a neural network whose size, and hence whose number of internal parameters, is optimal for modelling a given problem. While most systems are most efficiently modelled by networks specially designed for the application, the multi-layer perceptron family offers a good general learning tool for a wide range of applications. Since we do not know in advance what size to use, and because we usually need a larger network to learn a mapping than to implement a known solution, it is sensible to allow a small network to grow during early training. When a reasonable solution is found, it can be optimised during later training to give a small, fast and efficient network which is an accurate system model.

There are a variety of constructive algorithms that can add new nodes at suitable times. Upstart appears to be the best for binary mappings, while the best for

real-valued mappings is Cascade correlation. Both of these have the problem of long propagation delays from the network inputs to the outputs, although this is solved for Upstart by transforming the trained network to a single layer.

It is perhaps preferable for a network to maintain its single layer structure throughout construction, and some progress has been made in this area. Ash's dynamic node creation identifies a useful criterion for when to add nodes, and Hanson's Meiosis networks incorporate a method for determining the magnitude of divergence required for splitting a node in two. Hanson's proposed method does not, however, determine the direction of the split correctly, since the mean weights of new nodes are determined by perturbation in random directions. These principles are a starting point for a new split mechanism discussed in the next chapter.

We have seen that a measure of the sensitivity of the network to the presence of a weight or node forms a good criterion for determining whether it can be removed, and have suggested that at the other end of the scale, if applied to nodes, a suitably designed sensitivity measure might be used as a criterion for splitting a node in two. This appears to offer the basis for an integrated system for building networks, incorporating construction in early training and pruning as a solution is reached, enabling an optimal architecture to be found.